Amendments to the Claims

This listing of claims will replace all prior versions, and listings, of claims in the application:

Listing of Claims:

1. (Presently Amended) A computer system that employs a plurality of threads of execution to perform a parallel-execution operation in which the threads identify tasks dynamically and in which the computer system comprises:

A) a mechanism that associates a separate status-word field with each of the threads; and

B) a mechanism that operates the threads in a manner that ~~each thread~~:

i) each thread executes a task-finding routine to find tasks previously identified dynamically and performs tasks thereby found, with its associated status-word field containing a value indicating it is active, until the task-finding routine finds no more tasks;

ii) when the task-finding routine executed in step (i) finds no more tasks, that thread sets the contents of its associated status-word field to a value indicating it is inactive;

iii) after completing step (ii) and while the status-word field associated with any other thread contains a value indicating that the other thread is active, that thread ~~searches~~ continues to search for a task using the task-finding routine, and, if it finds one, sets its associated status-word field contents to a value indicating that it is active before attempting to execute a found task; and

iv) ~~if~~ during step (iii) when none of the status-word fields associated with other threads contains a value indicating that an associated thread is active and no task has been found, that thread terminates its performance of the parallel-execution operation.

2. (Original) A computer system as defined in claim 1 wherein the parallel-execution operation is a garbage-collection operation.

3. (Currently Amended) A computer system as defined in claim 1 wherein:

A) each thread has associated with it a respective work queue in which it places task identifiers of tasks that thread identifies dynamically;

B) the task-finding routine executed by ~~an executing~~ that thread includes performing an initial search for a task identifiers in the work queue associated with ~~the executing~~ that thread and, if that work queue contains no task identifiers that ~~the executing~~ thread can claim, thereafter performing a further search for a task identifier in at least one other task-storage location.

4. (Original) A computer system as defined in claim 3 wherein the parallel-execution operation is a garbage-collection operation.

5. (Original) A computer system as defined in claim 3 wherein the at least one other task-storage location includes at least one work queue associated with a thread other than the executing thread.

6. (Original) A computer system as defined in claim 5 wherein:

A) there is a size limit associated with each work queue;

B) when a given thread dynamically identifies a given task that would cause the number of task entries in the work queue associated with the given thread to exceed the size limit if a task identifier that identifies it were placed in that work queue, the given thread instead places that task identifier in an overflow list instead of in that work queue; and

C) the at least one other task-storage location includes at least one such over flow list.

7. (Original) A computer system as defined in claim 5 wherein the task-finding routine includes selecting in a random manner the at least one work queue associated with a thread other than the executing thread.

8. (Original) A computer system as defined in claim 5 wherein the further search includes repeatedly searching a work queue associated with a thread other than the executing thread until the executing thread thereby finds a task or has performed a number of repetitions equal to a repetition limit greater than one.

9. (Original) A computer system as defined in claim 8 wherein the task-finding routine includes selecting in a random manner the at least one work queue associated with a thread other than the executing thread.

10. (Original) A computer system as defined in claim 3 wherein:

A) there is a size limit associated with each work queue;

B) when a given thread dynamically identifies a given task that would cause the number of task entries in the work queue associated with the given thread to exceed the size limit if a task identifier that identifies it were placed in that work queue, the given thread instead places that task identifier in an overflow list instead of in that work queue; and

C) the at least one other task-storage location includes at least one such over flow list.

11. (Presently Amended) A computer system as defined in claim 1 wherein the contents of all of the status-word fields~~, when taken together, form a status word that fits~~ fit in a memory location accessible in a single machine instruction.

12. (Original) A computer system as defined in claim 11 wherein the parallel-execution operation is a garbage-collection operation.

13. (Original) A computer system as defined in claim 11 wherein each status-word field is a single-bit field.

14. (Previously Presented) A computer system as defined in claim 13 wherein each single-bit field contains a logic one to indicate that the associated thread is active and contains a logic zero to indicate that the associated thread is inactive.

15. (Presently Amended) For employing a plurality of threads of execution to perform a parallel-execution operation in which the threads identify tasks dynamically, a method comprising:

A) associating a separate status-word field with each of the threads; and

B) operating the threads in a manner that each thread:

i) each thread executes a task-finding routine to find tasks previously identified dynamically and performs tasks thereby found, with its associated status-word field containing a value indicating it is active, until the task-finding routine finds no more tasks;

ii) when the task-finding routine executed in step (i) finds no more tasks, that thread sets the contents of its associated status-word field to a value indicating it is inactive;

iii) after completing step (ii) and while the status-word field associated with any other thread contains a value indicating that the other thread is active, that thread ~~searches~~ continues to search for a task using the task-finding routine, and, if it finds one, sets its associated status-word field contents to a value indicating that it is active before attempting to execute a found task; and

iv) ~~if~~ during step (iii) when none of the status-word fields associated with other threads contains a value indicating that an associated thread is active and no task has been found, that thread terminates its performance of the parallel-execution operation.

16. (Original) A method as defined in claim 15 wherein the parallel-execution operation is a garbage-collection operation.

17. (Currently Amended) A method as defined in claim 15 wherein:

A) each thread has associated with it a respective work queue in which it places task identifiers of tasks that thread identifies dynamically;

B) the task-finding routine executed by ~~an executing~~ that thread includes performing an initial search for a task identifiers in the work queue associated with ~~the executing~~ that thread and, if that work queue contains no task identifiers that ~~the executing~~ thread can claim, thereafter performing a further search for a task identifier in at least one other task-storage location.

18. (Original) A method as defined in claim 17 wherein the parallel-execution operation is a garbage-collection operation.

19. (Original) A method as defined in claim 17 wherein the at least one other task-storage location includes at least one work queue associated with a thread other than the executing thread.

20. (Original) A method as defined in claim 19 wherein:

A) there is a size limit associated with each work queue;

B) when a given thread dynamically identifies a given task that would cause the number of task entries in the work queue associated with the given thread to exceed the size limit if a task identifier that identifies it were placed in that work queue, the given thread instead places that task identifier in an overflow list instead of in that work queue; and

C) the at least one other task-storage location includes at least one such over-flow list.

21. (Original) A method as defined in claim 19 wherein the task-finding routine includes selecting in a random manner the at least one work queue associated with a thread other than the executing thread.

22. (Original) A method as defined in claim 19 wherein the further search includes repeatedly searching a work queue associated with a thread other than the executing thread until the executing thread thereby finds a task or has performed a number of repetitions equal to a repetition limit greater than one.

23. (Original) A method as defined in claim 22 wherein the task-finding routine includes selecting in a random manner the at least one work queue associated with a thread other than the executing thread.

24. (Original) A method as defined in claim 17 wherein:

A) there is a size limit associated with each work queue;

B) when a given thread dynamically identifies a given task that would cause the number of task entries in the work queue associated with the given thread to exceed the size limit if a task identifier that identifies it were placed in that work queue, the given thread instead places that task identifier in an overflow list instead of in that work queue; and

C) the at least one other task-storage location includes at least one such over-flow list.

25. (Presently Amended) A method as defined in claim 15 wherein the contents of all of the status-word fields~~, when taken together, form a status word that fits~~ fit in a memory location accessible in a single machine instruction.

26. (Original) A method as defined in claim 25 wherein the parallel-execution operation is a garbage-collection operation.

27. (Original) A method as defined in claim 25 wherein each status-word field is a single-bit field.

28. (Previously Presented) A method as defined in claim 27 wherein each single-bit field contains a logic one to indicate that the associated thread is active and contains a logic zero to indicate that the associated thread is inactive.

29. (Presently Amended) A storage medium containing instructions readable by a computer system to configure the computer system to employ a plurality of threads of execution to perform a parallel-execution operation in which the threads identify tasks dynamically and in which the computer system comprises:

A) a mechanism that associates a separate status-word field with each of the threads; and

B) a mechanism that operates the threads in a manner that ~~each thread~~:

i) each thread executes a task-finding routine to find tasks previously identified dynamically and performs tasks thereby found, with its associated status-word field containing a value indicating it is active, until the task-finding routine finds no more tasks;

ii) when the task-finding routine executed in step (i) finds no more tasks, that thread sets the contents of its associated status-word field to a value indicating it is inactive;

iii) after completing step (ii) and while the status-word field associated with any other thread contains a value indicating that the other thread is active, that thread ~~searches~~ continues to search for a task using the task0finding routine, and, if it finds one, sets its associated status-word field contents to a value indicating that it is active before attempting to execute a found task; and

iv) if none of the status-word fields associated with other threads contains a value indicating that an associated thread is active, terminates its performance of the parallel-execution operation.

30. (Original) A storage medium as defined in claim 29 wherein the parallel-execution operation is a garbage-collection operation.

31. (Currently Amended) A storage medium as defined in claim 29 wherein:

A) each thread has associated with it a respective work queue in which it places task identifiers of tasks that thread identifies dynamically;

B) the task-finding routine executed by ~~an executing~~ that thread includes performing an initial search for a task identifiers in the work queue associated with ~~the executing~~ that thread and, if that work queue contains no task identifiers that ~~the executing~~ thread can claim, thereafter performing a further search for a task identifier in at least one other task-storage location.

32. (Original) A storage medium as defined in claim 31 wherein the parallel-execution operation is a garbage-collection operation.

33. (Original) A storage medium as defined in claim 31 wherein the at least one other task-storage location includes at least one work queue associated with a thread other than the executing thread.

34. (Original) A storage medium as defined in claim 33 wherein:

A) there is a size limit associated with each work queue;

B) when a given thread dynamically identifies a given task that would cause the number of task entries in the work queue associated with the given thread to exceed the size limit if a task identifier that identifies it were placed in that work queue, the given thread instead places that task identifier in an overflow list instead of in that work queue; and

C) the at least one other task-storage location includes at least one such over flow list.

35. (Original) A storage medium as defined in claim 33 wherein the task-finding routine includes selecting in a random manner the at least one work queue associated with a thread other than the executing thread.

36. (Original) A storage medium as defined in claim 33 wherein the further search includes repeatedly searching a work queue associated with a thread other than the executing thread until the executing thread thereby finds a task or has performed a number of repetitions equal to a repetition limit greater than one.

37. (Original) A storage medium as defined in claim 36 wherein the task-finding routine includes selecting in a random manner the at least one work queue associated with a thread other than the executing thread.

38. (Original) A storage medium as defined in claim 31 wherein:

A) there is a size limit associated with each work queue;

B) when a given thread dynamically identifies a given task that would cause the number of task entries in the work queue associated with the given thread to exceed the size limit if a task identifier that identifies it were placed in that work queue, the given thread instead places that task identifier in an overflow list instead of in that work queue; and

C) the at least one other task-storage location includes at least one such over flow list.

39. (Presently Amended) A storage medium as defined in claim 29 wherein the contents of all of the status-word fields~~, when taken together, form a status word that fits~~ fit in a memory location accessible in a single machine instruction.

40. (Original) A storage medium as defined in claim 39 wherein the parallel-execution operation is a garbage-collection operation.

41. (Original) A storage medium as defined in claim 39 wherein each status-word field is a single-bit field.

42. (Previously Presented) A storage medium as defined in claim 41 wherein each single-bit field contains a logic one to indicate that the associated thread is active and contains a logic zero to indicate that the associated thread is inactive.

43. (Presently Amended) A computer signal representing a sequence of instructions that, when executed by a computer system, configures the computer system to employ a plurality of threads of execution to perform a parallel-execution operation in which the threads identify tasks dynamically and in which the computer system comprises:

A) a mechanism that associates a separate status-word field with each of the threads; and

B) a mechanism that operates the threads in a manner that ~~each thread~~:

i) each thread executes a task-finding routine to find tasks previously identified dynamically and performs tasks thereby found, with its associated status-word field containing a value indicating it is active, until the task-finding routine finds no more tasks;

ii) when the task-finding routine executed in step (i) finds no more tasks, that thread sets the contents of its associated status-word field to a value indicating it is inactive;

iii) after completing step (ii) and while the status-word field associated with any other thread contains a value indicating that the associated thread is active, that thread ~~searches~~ continues to search for a task using the task-finding routine, and, if it finds one, sets its associated status-word field contents to a value indicating that it is active before attempting to execute a found task; and

iv) ~~if~~ during step (iii) when none of the status-word fields contains a value indicating that an associated thread is active and no task has been found, that thread terminates its performance of the parallel-execution operation.

44. (Original) A computer signal as defined in claim 43 wherein the parallel-execution operation is a garbage-collection operation.

45. (Currently Amended) A computer signal as defined in claim 43 wherein:

A) each thread has associated with it a respective work queue in which it places task identifiers of tasks that thread identifies dynamically;

B) the task-finding routine executed by ~~an executing~~ that thread includes performing an initial search for a task identifiers in the work queue associated with ~~the executing~~ that thread and, if that work queue contains no task identifiers that ~~the executing~~ thread can claim, thereafter performing a further search for a task identifier in at least one other task-storage location.

46. (Original) A computer signal as defined in claim 45 wherein the parallel-execution operation is a garbage-collection operation.

47. (Original) A computer signal as defined in claim 45 wherein the at least one other task-storage location includes at least one work queue associated with a thread other than the executing thread.

48. (Original) A computer signal as defined in claim 47 wherein:

A) there is a size limit associated with each work queue;

B) when a given thread dynamically identifies a given task that would cause the number of task entries in the work queue associated with the given thread to exceed the size limit if a task identifier that identities it were placed in that work queue, the given thread instead places that task identifier in an overflow list instead of in that work queue; and

C) the at least one other task-storage location includes at least one such over flow list.

49. (Original) A computer signal as defined in claim 47 wherein the task-finding routine includes selecting in a random manner the at least one work queue associated with a thread other than the executing thread.

50. (Original) A computer signal as defined in claim 47 wherein the further search includes repeatedly searching a work queue associated with a thread other than the executing thread until the executing thread thereby finds a task or has performed a number of repetitions equal to a repetition limit greater than one.

51. (Original) A computer signal as defined in claim 50 wherein the task-finding routine includes selecting in a random manner the at least one work queue associated with a thread other than the executing thread.

52. (Original) A computer signal as defined in claim 45 wherein:

A) there is a size limit associated with each work queue;

B) when a given thread dynamically identifies a given task that would cause the number of task entries in the word queue associated with the given thread to exceed the size limit if a task identifier that identifies it were placed in that work queue, the given thread instead places that task identifier in an overflow list instead of in that work queue; and

C) the at least one other task-storage location includes at least one such over flow list.

53. (Presently Amended) A computer signal as defined in claim 43 wherein the contents of all of the status-word fields~~, taken together, form a status word that fits~~ fit in a memory location accessible in a single machine instruction.

54. (Original) A computer signal as defined in claim 53 wherein the parallel-execution operation is a garbage-collection operation.

55. (Original) A computer signal as defined in claim 53 wherein each status-word field is a single-bit field.

56. (Previously Presented) A computer signal as defined in claim 55 wherein each single-bit field contains a logic one to indicate that the associated thread is active and contains a logic zero to indicate that the associated thread is inactive.

57. (Presently Amended) A computer system that employs a plurality of threads of execution to perform a parallel-execution operation in which the threads identify tasks dynamically, the computer system including:

A) means for associating a separate status-word field with each of the threads; and

B) means for operating the threads in a manner that ~~each thread~~:

i) each thread executes a task-finding routine to find tasks previously identified dynamically and performs tasks thereby found, with its associated status-word field containing a value indicating it is active, until the task-finding routine finds no more tasks;

ii) when the task-finding routine executed in step (i) finds no more tasks, that thread sets the contents of its associated status-word field to a value indicating it is inactive;

iii) after completing step (ii) and while the status-word field associated with any other thread contains a value indicating that the other thread is active, that thread ~~searches~~ continues to search for a task using the task-finding routine, and, if it finds one, sets the status-word field contents to the activity-indicating a value indicating that it is active before attempting to execute a found task; and

iv) ~~if~~ during step (iii) when none of the status-word fields associated with other threads contains a value indicating that an associated thread is active and no task has been found, that thread terminates its performance of the parallel-execution operation.